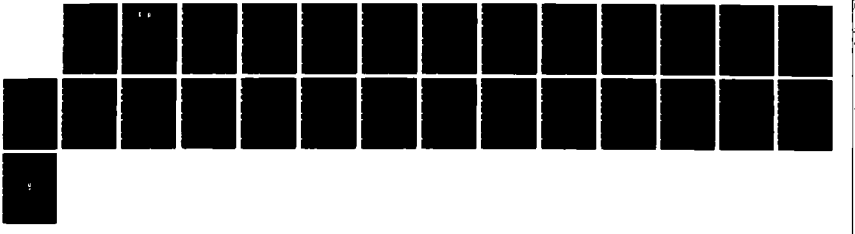
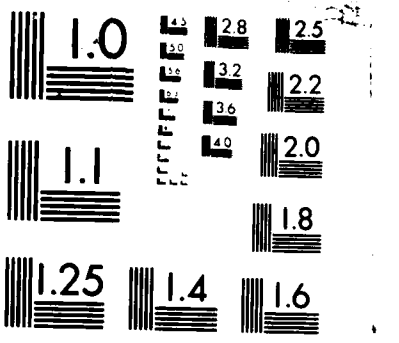
AD-A177 311 SEARCH ALGORITHMS AND THEIR IMPLEMENTATION(U) DUKE UNIV 1/1
DURHAM NC DEPT OF COMPUTER SCIENCE D W LOVELAND
29 AUG 86 AFOSR-TR-87-0237 AFOSR-81-0221

UNCLASSIFIED F/G 12/2 NL






MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS 1963-A

SECURITY CLASSIFICATION OF THIS PAGE

(2)

REPORT DOCU

# AD-A177 311

DTIC ELECTE MAR 0 2 1987 S D

1a REPORT SECURITY CLASSIFICATION

2a SECURITY CLASSIFICATION AUTHORITY

2b DECLASSIFICATION/DOWNGRADING SCHEDULE

3 DISTRIBUTION/AVAILABILITY OF REPORT

Approved for public release; distribution unlimited.

4 PERFORMING ORGANIZATION REPORT NUMBER(S)

5 MONITORING ORGANIZATION REPORT NUMBER(S)

AFOSR-TR 87-0237

6a NAME OF PERFORMING ORGANIZATION

Duke University
Computer Science Department

6b OFFICE SYMBOL (If applicable)

7a NAME OF MONITORING ORGANIZATION

Air Force Office of Scientific Research

6c ADDRESS (City, State and ZIP Code)

Durham, NC 27706

7b ADDRESS (City, State and ZIP Code)

Air Force System Command
Bolling Air Force Base
Washington, D.C. 20332

8a NAME OF FUNDING/SPONSORING ORGANIZATION

AFOSR

8b OFFICE SYMBOL (If applicable)

NM

9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

Grants AFOSR-81-0221, AFOSR-83-0205

8c ADDRESS (City, State and ZIP Code)

Bolling AFB DC 20332-6448

10 SOURCE OF FUNDING NOS

| PROGRAM ELEMENT NO | PROJECT NO. | TASK NO. | WORK UNIT NO |
|---|---|---|---|
| 61102F | 2304 | A7 | |

11 TITLE (Include Security Classification)

Search Algorithms and Their Implementation

12 PERSONAL AUTHOR(S)

Donald W. Loveland

| 13a TYPE OF REPORT | 13b TIME COVERED | 14 DATE OF REPORT (Yr., Mo., Day) | 15 PAGE COUNT |
|---|---|---|---|
| Final Scientific Report | FROM 6/81 TO 6/86 | 1986, Aug. 29 | 24 |

16 SUPPLEMENTARY NOTATION

| 17 COSATI CODES | | | 18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB GR | Final Scientific Report |
| | | | |
| | | | |

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Research performed under grants AFOSR-81-0221 and AFOSR-83-0205 is summarized. The focus was on search algorithms and their implementation; the research spanned topics from studies of minimax search trees to searching for test-and-treatment decision trees of low cost. Among our more important contributions are methods for use of expectation to correct speech input, the embedding of associative networks in massively parallel computers, and algorithms for computing rule strenghts in certain rule-based expert system architectures. Eleven projects receiving support from these grants are summarized.

DTIC FILE COPY

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT

UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS ☐

21 ABSTRACT SECURITY CLASSIFICATION

22a NAME OF RESPONSIBLE INDIVIDUAL

~~D. W. Loveland~~ Capt Crowley

22b TELEPHONE NUMBER (Include Area Code)

~~919-684-3048~~ 767-5025

22c OFFICE SYMBOL

NM

DD FORM 1473, 83 APR

EDITION OF 1 JAN 73 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE

2304 C1

# SEARCH ALGORITHMS AND THEIR IMPLEMENTATION

AFOSR-TR- 87-0287

Donald W. Loveland
Department of Computer Science
Duke University
Durham, N.C. 27706





29 August 1986

Final Scientific Report
Grants AFOSR-81-0221, AFOSR-83-0205

Distribution Statement

Prepared for
Air Force Office of Scientific Research
Bolling Air Force Base
Washington, DC 20332

Table of Contents

**Abstract**. Research performed under grants AFOSR-81-0221 and AFOSR-83-0205 is summarized. The focus was on search algorithms and their implementation; the research spanned topics from studies of minimax search trees to searching for test-and-treatment decision trees of low cost. Among our more important contributions are methods for use of expectation to correct speech input, the embedding of associative networks in massively parallel computers, and algorithms for computing rule strengths in certain rule-based expert system architectures. Eleven projects receiving support from these grants are summarized.

## Introduction

This is the final report for Grants AFOSR-81-0221 and AFOSR-83-0205, from June 1981 to June 1986. The grants, entitled "Search Algorithms and their Implementation", supported general and wide-ranging studies in search methodologies and applications. The resulting research indeed covered many aspects of search, from the classical artificial intelligence studies of minimax tree search to searching for test-and-treatment decision trees of low cost, and included search with very limited resources, searches for learnable functions in a full function space, and search for "critical sets" (boundary sets) of monotonic boolean set functions. Other studies include expert systems, where the searching was over rule sets. With the exception of building an expert system for the learning experience our research in expert systems focussed on research topics more associated with search than usual for expert systems. We studied aiding the expert in his search for negative consequences when a new rule is added to a rule-based system, and the question of automating the assignment of certainty values in a rule-based inference network. Research in natural language understanding led to use of expectation in speech understanding and a method for effectively embedding semantic nets in a massively parallel computer.

Much was accomplished over a range of research subtopics within the search methodology and implementation framework. We present summaries of the individual studies on the following pages. After each summary title we list the supported investigators for that project. Non-supported personnel, often major contributors, often participated in projects. A primary example is Gary Jackoway, the major contributor to the work on associative networks, who was supported by Hewlett-Packard.

To highlight work that we think is particularly important we mention the use of expectation to correct speech input, the embedding of associative networks in a parallel architecture, and the algorithms for computing rule strengths in certain rule-based expert systems architectures. The critical set algorithm already has found one application outside its original use and may prove to be of general importance. The work in non-minimax search procedures should be of great interest to people studying search theory. We are excited about what has been learned regarding representation capability of learning machines and feel that this should influence the approach to designing learning machines as well as our expectations for such machines. Developing an integrated theory, including practical algorithms, that tells us how to select low-cost procedures to detect *and treat* defective objects when they are immersed in a otherwise functioning environment is clearly important and we have made initial progress on this difficult problem. We have developed a deep understanding of some of the complex structure of this problem and work will continue that will utilize this knowledge in more general cases than completed to date.

The order of presentation, largely arbitrary, is grouped by subtopic. Some of the projects that have not had reports or theses completed to date have extensive entries in this report, with the largest entry the study of test-and-treatment procedures because of its importance and our delay in completing the basic technical report on approximate solutions. (The report was slated for completion this summer, and will be completed by the year's end.) Also, several reports have been completed on this subject. They deal with the dynamic programming solution to the problem, in terms of parallel computation methods and a quickly computed subcase. The major report is to help readers of this summary report understand the structure and complexities of the test-and-treatment problem.

Eleven papers have already appeared or are accepted for publication with one other already submitted. Three other papers are in active preparation and should be submitted to journals by the end of the year. Two Ph.D. dissertations and a Masters' thesis have been written with a third Ph.D. dissertation close to conclusion. A number of conference presentations have been given on topics supported by these grants. Although it is hard to measure the ultimate direct value of this research to the Air Force, one

immediate contribution is a Ph.D. (D. Mutchler) who now works at the adjacent Naval Research Labs in Washington.

It is with gratitude for the Air Force support of this basic research that we submit our final scientific report.

**The correction of ill-formed input using history-based expectation with applications to speech understanding (Fink, Biermann).**

A method for correction of ill-formed input was developed that acquires dialogue patterns in typical usage and uses these patterns to predict new inputs. Error correction is done by strongly biasing parsing toward expected meanings unless clear evidence from the input shows the current sentence is not expected. A dialogue acquisition and tracking algorithm was developed and its performance was studied in an implementation of a voice interactive system. A series of tests were made to show the power of the error correction methodology when stereotypic dialogue occurs.

**Associative networks on a massively parallel computer (Biermann, Wagner).**

Many natural language projects in the past fifteen years have used semantic networks as their underlying knowledge representation. In a separate realm recent breakthroughs in very large scale integration (VLSI) have led to designs for machines with vast numbers of processors. In this work we were able to marry these two technologies. A generalization of semantic networks, called an associative network, was mapped onto a massively parallel processor which is currently under development. The results show:

The time required to process a query is dependent strictly on the pattern of the query, not on the size of the classes being processed. A system built using this knowledge representation will give consistent semantic processing performance.

The order of processing a query does not affect the speed. Thus there is no need for heuristics and monitors to determine the most efficient way to process a query.

Although we do not receive anywhere near an n-fold speedup by using n processors, we still receive significant performance benefits over a single processor.

The associative network may be used not just as a semantic network; for example, it also allows some problems involving numerical minimizations to be solved efficiently.

The primary result of this work is that a large number of simple processors, each responsible for a small piece of information, can work in unison to answer queries significantly faster than a single, highly complex processor can. The work is reported in G. Jackoway's A.M. thesis and a paper by Jackoway is under revision in preparation for submission to publication.

**Signature table systems and learning (Biermann).**

A characterization theorem was developed for the classes of functions which are represented by signature table systems. The usefulness of the theorem was demonstrated in the analysis and synthesis of such systems. The limitations on the power of these systems come from the restrictions on the table alphabet sizes, and a technique was found for evaluating these limitations. A practical learning system was proposed and analyzed in terms of the theoretical model. Then an improved method was developed and studied in a series of experiments.

**On the errors that learning machines will make (Biermann, Gilbert, Fahmy).**

A learning model will be discussed where binary function f of p binary variables $x_1, x_2, \ldots, x_p$ is to be learned from example input-output behaviors. At each time t = 1,2,3,..., the learning machine receives a sample input-output pair for the target function and guesses what that target function is.

Most learning machines are capable of guessing (or learning) only L functions where L is less than the set of all possible functions $2^{2^p}$. There are advantages to having L large: more functions can be learned, and in general, when the target function is not precisely learnable there will be a learnable function not too distant from that target function. Thus the learning machine will be able to choose a function which agrees with the target function on most inputs even though it will be in error on some inputs. There are also advantages to having L small if the problems with error are not too severe: learning can occur much more quickly if there are fewer learnable functions to choose from. This paper is concerned with a number of different learning machines, their associated values for L and the nature of the trade-off between having large L and little expected error versus having small L and short expected learning time.

For example, the signature table learning model of Arthur Samuel was studied. A characterization of the class of learnable functions was found which gives insight into how the mechanism works and what types of functions it can acquire. The characterization specifies the form that certain matrices of function values must have in order for the function to be realized. This leads to a methodology for computing L and estimating the expected error that signature table systems will have in attempting to learn a randomly selected target function from the set of all possible functions.

Other learning models have similarly been studied such as the linear evaluation systems, the Boolean conjunctive normal form learning methodology of Valiant, and "truncation machines" which simply memorize the outputs with the assumption that they are determined by a specified subset of the inputs.

The L learnable functions for a given machine may be widely spread across the space of all possible functions so that every possible function is near, using Hamming distance as a measure, some learnable function. They may also be very poorly scattered so that some possible target functions are very far from any learnable function. In order to gather information regarding the quality of these learnable function distributions, a new learning machine was invented, the "G-machine", which spreads its learnable behaviors in a near optimal fashion. The G-machine thus can learn with very low expected error for a given value of L and serves as a standard for comparison with other learning machines. The general result in some simulations was that most learning machines achieved expected errors which were surprisingly close to the best known values.

**Knowledge evaluation (Loveland, Valtorta).**

In August 1983 we presented a paper at IJCAI-83 in Karlesruhe on the type of subject matter we mean by "knowledge evaluation" within the expert system domain. The paper "Detecting ambiguity: an example of knowledge evaluation" developed an aid for testing the compatibility of a new rule added to a rule base for an expert system of the classification type. (Classification expert systems include many of those doing diagnosis, such as MYCIN.) When a new rule is added, a great deal of testing is sometimes necessary to determine the compatability of the new rule with the rest of the knowledge base. (Note that in expert systems it is not reasonable to *prove* the compatibility of the rule base because the problem does not have clear formal specifications. If it did there would probably be little use for an expert system.) The method proposed reduced greatly the number of input vectors needed to test for compatibility.

**Graduate Course Adviser (Valtorta, Loveland).**

The Graduate Course Adviser (GCA) is an expert system to advise graduate students regarding course selection, built as a learning tool to help us understand the techniques and problems of developing expert systems. It is a rule-based system, initially patterned after MYCIN (see [1]), which has evolved into a multi-stage system using algorithms and tables as well as rules. Although our main research in

expert systems lies elsewhere, lessons learned in the design of GCA have led to two "invited" papers to be given in workshops on knowledge-based systems ([2],[3]). We received volunteer help from Bruce Smith, a Ph.D. candidate at the University of North Carolina, and from Tim Harrison, a Duke graduate student.

Because our work in knowledge evaluation has been viewed as "too theoretical" by some practitioners in the knowledge-based systems field, we have felt it wise to establish our credentials by gaining some experience as practitioners. Most important, of course, is the actual experience itself, helping us to judge what will be needed in future knowledge-based systems.

[1]. Shortliffe, E.H. *MYCIN: Computer-based Medical Consultations.* Amer. Elsevier, New York, 1976.

[2]. Valtorta, M. Knowledge refinement in rule bases for expert systems: an application-driven approach. *First Int'l Workshop on Expert Database Systems,* Kiawah Island, SC, Oct. 1984.

[3] Valtorta, M., B. Smith and D.W. Loveland. The Graduate Course Adviser: a multiphase rule-based expert system. *IEEE Workshop on Principles of Knowledge-Based Systems.* Denver, Dec. 1984.

**Critical sets (Loveland, Lanskron).**

Our previous work on knowledge evaluation led to the discovery of a useful algorithm that surprisingly does not seem to have appeared in the literature. This was ascertained not only by a brief literature search but by asking those in the field most likely to know. Although none knew of any previous occurrence of the algorithm in the literature, all expressed surprise that it seemed not to be there. Several even contributed interesting approaches to the problem, including the referee of our first submission in this regard. With these new ideas in hand we discovered that there were several approaches to the problem, and that our original discovery appeared to remain the most attractive but there were close runner-up algorithms. Overall, the clarity of the algorithms was greatly improved by this second go-around. The paper, *Finding Critical Sets,* is approved for publication in the *Journal of Algorithms.*

A critical set is defined in terms of a binary monotone set function, that is, a function $f$ defined over all subsets of a universe $U$ taking only values 0 and 1 and satisfying $f(S)=0$ if $f(T)=0$ and $S \subseteq T$. A set $S$ is critical iff $f(T) = 1$ for all $T$ such that $S \subseteq T$ and $f(R) = 0$ for all $R$ such that $R \subset S$. Upper bounds on the worst case times for the algorithms, determined by the number of calls to the binary set function, vary downwards from $2r\lceil lg_2\, n \rceil$ function calls. Here $n$ is the size of $U$ and $r$ is the size of the critical set found. We find that the conceptually easiest algorithm is not the easiest to program and also that the algorithms with better upper bounds fare more poorly in practice, based on a small sample of random trials. Although finding one critical set is easy we also show that it can quickly get very difficult to find additional critical sets, simply because it can be hard to separate known critical sets from further possible critical sets.

The initial application occurred in a proposed method for aiding the expert to reduce the number of tests needed to check induced ambiguities when a new inference rule was introduced into a classification-type rule-based system. ( The application was reported in the paper *Detecting ambiguities; an example of knowledge evaluation* given at the 1983 International Joint Conference of AI.) Since then a new application of critical sets has been found by Mr. Valtorta in work on computing attenuations. The bottom line is that the critical set algorithms are likely to be useful in a number of settings.

**Expert systems: computing rule weights (Valtorta, Loveland).**

For several years we have had an interest in an area we call *knowledge evaluation* which is concerned with finding ways of assisting the user in maintaining and judging the quality of a knowledge base, presumably a large knowledge base in existence for a number of years. One question we became interested in along this line is determining or adjusting rule weights automatically, so as to make a rule-based system more accurately perform its task. We call these the *synthesis* and *refinement* problems, respectively. Although the ultimate ideal is to have the system learn the entire rule space from tests (examples) it is a

much more modest endeavor to have the rule weight (the "degree of validity or worth" of the rule) established by testing on known cases. It is a natural intermediate step because experts may well know a particular rule-of-thumb but be very unsure how much weight to give the rule outcome in interaction with the other rules of the system. In particular, correlations between rules are very hard to judge; for example, a rule may seem very much worth adding in isolation but in fact nearly be covered by other rules already in the system.

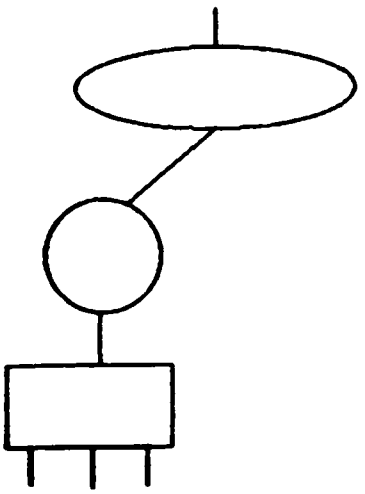
Our approach to this study is different than that of usual learning algorithms. We also are interested in algorithms that would allow us to compute directly the strengths of the rules, which we call *attenuations* because the strength of the inference made by the rule is attenuated (reduced) by the amount specified. The class of attenuations that we have considered in our models is that of multiplicative weights. (A weight of 1 would mean no attenuation.) That is, we wish to understand how difficult it is, and find algorithms where appropriate, to use the information from a block of tests *in any fashion*, not just incrementally, to determine rule strength weights, or attenuations. Perhaps surprisingly, we find that such determinations can be easy in certain cases, although in many cases it is provably difficult almost certainly (i.e., NP-Hard) to compute such attenuations.

The knowledge-based systems that we consider operate by chaining rules together. For most of our studies a *rule* has the form IF (P1 & P2 & ... & Pn) THEN C WITH ATTENUATION a, where P1, P2,...,Pn, and C are weighted propositions and a is a real number between 0 and 1 inclusive. By *weighted proposition* we mean a statement, possibly true or false, with a weight, called a certainty factor (CF), that reports the degree of belief that the associated proposition is true. CF's have values between 0 and 1, inclusive. A *combinator* is a function from a vector of CF's to a CF that assigns a single number to the conjunction P1 & P2 &...& Pn of premises Pi of the rule. The most used function here is the MIN function. For our purpose, we multiply the combinator output by the attenuation value A to determine the CF associated with the conclusion C of the rule. Many rules may have the same conclusion; their collective input is merged by an *integrator* function which defines the CF value associated with weighted proposition C in the rule system. One function that is often used as an integrator is MAX, although others are frequently used. We have obtained some results for knowledge-based systems in which *probabilistic addition* is used in place of MAX. Probabilistic addition maps two positive integers x and y between 0 and 1 into x + (1-x)y.

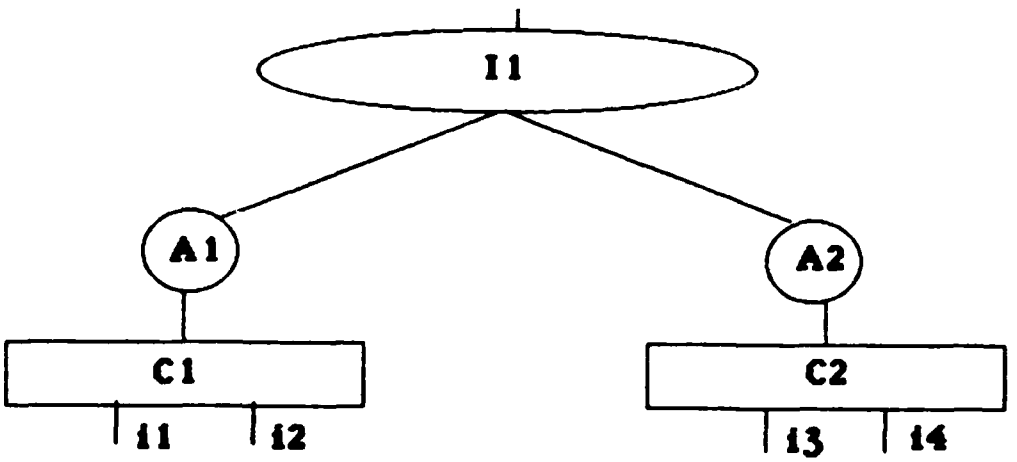
Although the simple if...then rule form we use here is very common and will suffice for the present discussion, we note that we have considered variations to it. In particular, some results we mention later consider the role played by *predicate functions*. A predicate function takes the CF of a weighted proposition in the premise of a rule and returns a CF for that same proposition. (An example is a function that maps CF values less than a threshold to 0 and leaves other values unchanged.)

We found it convenient to represent knowledge-based systems as graphs, called *inference nets*, composed of basic building blocks that represent a single rule. A typical basic subgraph representing a rule

with three weighted propositions in its premise is shown below:



where the oval, circle, and square represent an integrator, an attenuator, and a combinator, respectively. For example, the knowledge-based system composed of the two rules IF P1 & P2 THEN P5 WITH ATTENUATION a1 and IF P3 & P4 THEN P5 WITH ATTENUATION a2 would be represented as shown below.




The certainty factors of weighted propositions P1, P2, P3, and P4 flow from the input lines i1, i2, i3, and i4 through combinators C1 and C2, attenuators A1 and A2, and integrator I1. The output of integrator I1 is the certainty factor of weighted proposition P5.

Inference nets realize functions from vectors of CF's to vectors of CF's; a point in the graph is called a *test*. In the example above, the input vectors have cardinality 4, while the output vector consists of a single CF.

We assume that the inference net to be built is completely specified, except for the values of the attenuators. The designer of the knowledge-based system must determine or adjust (if estimates of the values are given) these values. We consider two ways in which the designer can learn the values. We call the first model of learning the *complete case* and the second model the *incomplete case*. In the complete case, all tests are available; in the incomplete case, only a limited collection of tests is available.

In the complete case, we postulate the existence of a *perfect expert*, i.e., an oracle that outputs the output part of a test, when given the input part of a test. For example, the perfect expert would be able to correctly diagnose whatever description of a patient were to be presented to it. The patient description is an assignment of certainty factors to a predetermined set of patient descriptors (weighted propositions), while the diagnosis is an assignment of certainty factors associated with a predetermined set of possible diseases (also weighted propositions). (For inference nets that are trees, this set would include only one disease.)

The results that we have obtained for the complete case can be summarized as follows:

(a) It appears easy to synthesize attenuations for *trees* with MIN for OR nodes, MAX for AND nodes and multiplicative attenuations. In particular, it is easy for real attenuations and real certainty factors, within the usual rounding errors introduced by computer multiplication.

(b) It is NP-Hard to synthesize attenuations for *trees* with MIN, MAX and multiplicative attenuations when the predicate functions are allowed to perform "sign flipping" of the certainty factors. (This is performed in MYCIN by the predicate function THOUGHTNOT.)

(c) It is NP-Hard to synthesize internal attenuations for *acyclic graphs* using MIN, MAX and multiplicative attenuations.

(d) It is easy to synthesize input attenuations for the same model as in (c).

(e) It is NP-Hard to synthesize attenuations for *chains* using MIN, MAX and some choices of attenuations that are not closed under composition.

(f) Similar results hold for probabilistic addition in place of MAX. In particular, it is easy to synthesize attenuations for trees with real multiplicative attenuations.

The results that we have obtained for the incomplete test case can be summarized as follows:

(a) Synthesis of attenuations is NP-Hard even for trees with MIN, MAX, multiplicative attenuations, bounded fan-in to the MIN's, and fixed depth of the tree. Note that this models well some typical rule-based expert systems: bounded number of premises to each rule (independent of the size of the rule base) and short inference chains, also independent of the size of the rule base. In particular, if determining attenuations is hard for this highly constrained model, then finding certainty factors for expert systems using MIN and MAX surely is hard, if done from test cases. We have proved that it is NP-Hard to determine attenuations in this restricted setting.

(b) As above, with probabilistic addition in place of MAX.

(c) Approximate attenuation synthesis. It is NP-Hard to synthesize attenuations which are within less than 50% (of the allowable range of attenuations) away from the correct ones.

(d) Heuristic algorithms. We have tried to exploit the concept of *influential input* for the MIN/MAX case with multiplicative attenuations. In this case, the output of the inference net is always equal to the attenuated value of one of the inputs. We have proved that the problem of setting attenuations remains NP-Hard for trees even when an influential input is specified for each test. We have noticed, however, that it is simple to synthesize attenuations if the tests are augmented to contain the certainty factors of intermediate hypothesis.

(e) Refinement from almost exact attenuations. We consider a family of trees as in (a), with estimates for all attenuations. We assume that the estimates are closer to the correct attenuations than a constant. This refinement problem is NP-Hard for any positive value of this constant, however small.

This work is contained in the doctoral dissertation of Marco Valtorta, which is nearing completion. Although Mr. Valtorta is now fully employed elsewhere, we hope for essential completion during a current leave of absence and completion of the thesis itself by the end of this calendar year.

**Search strategies for minimax trees (Ballard, Reibman).**

In our first work we showed how to modify the standard alpha-beta pruning stategy to deal with minimax trees into which chance nodes have been introduced (Ballard [1,2]). In the context of this work we defined the *value* of a chance node as the possibly-weighted *average* of its children's values. The idea behind this definition is to select a move whose payoff has the highest possible expected value. We showed by both empirical and closed-form analysis that our algorithm provides a considerable savings over the "obvious" algorithm (from 30% to as much as an order of magnitude).

We used the standard maximum and minimum of children's values as the back-up criterion for max and min nodes. We observe by analogy, however, that the "proper" goal for game tree search is *always* to choose the move with highest expected value. This suggests that when exhaustive search is not possible and/or our opponent is thought to be fallible, the values of even ordinary max and min nodes might profitably be defined as a suitably weighted average. Of course, the weight assigned to the best-looking child of a node, which in traditional analysis fully determines the parent value, will have a very high weight assigned to it, perhaps 0.9 or more. We further observe that this variant backup criterion for max and min nodes could just as well be applied to pure minimax trees as to the *-minimax trees we have previously studied. In fact, we maintain that the proper context in which to carry out an initial study of variant (i.e. non-minimax) strategies is that of pure minimax trees, so that the primary question becomes one of determining a strategy for assigning "suitable" weights to the children of a node. We note that this goal is one of improving the *performance* of the search algorithms, rather than merely their *efficiency*.

Based upon the observations above, we studied the probabilistic tree searching techniques with a new emphasis on non-minimax strategies for searching minimax trees. Our overall objectives were similar to those of previous studies (e.g. Slagle and Dixon [8]) but the model we used is as (a) *degrees of dependence* among the values of sister nodes; (b) the *relation of static values to the true values* of non-leaf nodes; and (c) ways of using information about *opponent strategy* to increase the expected outcome. In an attempt to make our results as widely applicable as possible, we have been especially careful to formulate a model in which all parameters can be varied independently of the others. We expect our work to add to the repertoire of probabilistic techniques used in economic and military modeling, and also to respond to various technical issues (e.g. the conditions under which search "pathology" occur) raised by Nau [6] and Pearl [7].

*Motivation.* We were interested in finding improved strategies for decision-making in the face of incomplete information, such as that arising for large minimax trees for which exhaustive search is infeasible. Since most minimax trees of interest are much too large to be searched exhaustively, heuristic search methods are required. The conventional way of operating in these situations is to (1) search as deeply as possible, (2) appeal to a "static" evaluation function at deep nodes which are being treated as leaves, and (3) somehow "backup" these static values toward the root of the tree. In practice, the standard minimax rules are usually used for backup, but there is no theoretical foundation for doing so. The matter is discussed in various forms by Nau [6], Slagle and Dixon [8], and Truscott [9]. The problem is that the minimax backup rules are valid only when they are being supplied with perfect information. It is not immediately clear how they function when approximations are being used.

*The Model.* In [3] we investigate ways of outperforming minimax. We were especially interested in formulating a realistic model of minimax trees, and also with being able to vary the individual parameters of the model independently. The tree model we have begun to work with incorporates both structural and semantic properties. The first few parameters involve the standard structural characterizations of the game p and portions of it being searched from move to move.

D - the depth of the initial game tree
B - the branching factor
M - the search depth of max
N - the search depth of min

The remaining parameters deal with semantic features which are much harder to deal with and which we feel are largely understood.

LAS - a "leaf assignment strategy" that assigns a value to each of the B**D leaves of the tree. The primary problem here is to capture dependencies among values of sister nodes realistically.

STAT - a way of assigning "static" values to nodes. The problem here is to control how well the true value of a node is predicted by its static value.

OPP - a model of the opponent. In general this might involve knowledge about his search depth, backup criteria, and static evaluation function. Since such detailed knowledge is seldom available, we have chosen to characterize the overall *quality* of the opponent's play by a composite value which we refer to as "playing strength".

Given the model parameters above, our goal was to determine optimal backup criteria for max. In particular, we attempt to estimate for each child of a node the probability that the opponent will select it. From these probabilities we can form a *weighted average* of the children's value as the value to be passed back. We refer to the new form of backup as the "*-Min" strategy. The computations of *-Min are similar to those for *chance* nodes as described in Ballard [2], but *-Min is differently motivated. One can see that *-Min is a generalization of minimax, which assumes a probability of 1 for the move that looks best to us and 0 for the rest. At the other extreme, we would characterize an opponent who makes random moves by assigning a probability of 1/B for each node, where B is the branching factor. Furthermore, one can show that in its simplest form, *-Min reduces to the "M and N" algorithm of Slagle and Dixon [8], though again the motivation is somewhat different.

We then extended the study to the more elaborate "D-PBU" search strategy we had formulated during the spring of 1982, which was motivated by the "product rule" methods proposed by Pearl which have subsequently been studied by him and Nau. Collecting together all the ideas and strategies proposed by ourselves, as well as recent proposals of Nau and Pearl and early work by Slagle and Dixon, we undertook a comprehensive series of empirical studies of several non-minimax strategies. The principal results of these studies are: (1) all non-minimax criteria studied yield a slight but significant improvement over minimax; (2) strategies based upon a simple average do better than those based on products; (3) the most likely, and in some cases the most important, opportunity to outperform minimax arises when minimax is faced with a tie; (4) departures from minimax tend to occur in unfavorable positions; and (5) non-minimax strategies exhibit uneven performance when pitted against one another. These results, along with a definition of the non-minimax strategies that were considered and an account of the historical relation among them also appear in [3].

A study of the role of opponent error was undertaken by Reibman, and a paper written [4] giving a model of player fallibility, and preliminary results on its expected performance.

References


[1] Ballard, B.W. A search procedure for perfect information games of chance: its formulation and analysis. *Second National Conference on Artificial Intelligence*, 1982, pp. 111-114.

[2] Ballard, B.W. The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence*, Oct. 1983, pp. 327-350.

[3] Ballard, B.W. Non-minimax search strategies for minimax trees: theoretical foundations and empirical studies. Conditionally accepted in *Artif. Intelligence*. To be merged with [4].

[4] Reibman, A., B. Ballard. Non-minimax search strategies for use against fallible opponents. *Third Nat'l Conf. on AI*, Aug. 1983. Cond. accepted in *Artif. Intelligence*. To be merged with [3].

[5] Fuller, S.H., J.G. Gaschnig, and J.J. Gillogly. An analysis of the alpha-beta pruning algorithm. Dept. of Computer Science Report, Carnegie-Mellon University, 1973.

[6] Nau, D. Pathology on game trees: a summary of results. *First National Conference on Artificial Intelligence*, 1980, pp. 102-104.

[7] Pearl, J. Game-searching theory: survey of recent results. *SIGART Newsletter* No. 80, 1982, pp. 70-75.

[8] Slagle, J.R. and J.K. Dixon. Experiments with the M & N tree-searching program. *Comm. ACM 13* (1970), 3, pp. 147-152.

[9] Truscott, T.R. Minimum variance tree searching. *First Int. Symposium on Policy Analysis and Information Systems*, 1979, pp. 203-209.

**Search with limited resources (Mutchler, Loveland).**

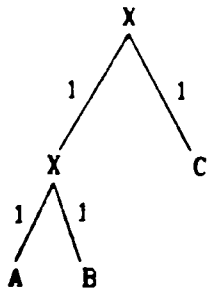
We summarize the problem(s) we have labeled "search with limited resources". The following questions provide motivation for studying search with limited resources:

*Where can search resources best be expended to gain "useful" information?*
*How should one use acquired information to gain more information?*
*Does knowing the limits of search resources help one search more effectively?*
*What algorithms are good approximations to the optimal search policy?*

To investigate these questions in an analytic framework, we use the following probabilistic model. We generate trees as Karp and Pearl [2] and others have done. That is, we consider complete binary depth $N$ trees. Arcs are independently assigned 1 or 0 with probability $p$ and $q = 1 - p$ respectively. Each leaf value is the sum of the arc values on the path from the root to the leaf. Hence values of sibling leaves are positively correlated. One argument for using the arc-sum model is that real-world search trees often possess this correlation property.

This model features a scout who searches such trees as Karp and Pearl [2] did. The scout begins at the root and *expands* nodes, thereby finding the values of the two arcs below it. No node can be expanded until its parent has been expanded. After expanding $n$ nodes, the scout must halt and report to the general. The general uses the scout's findings to select a leaf of the tree. The general's goal is to minimize the expected value of the chosen leaf. The questions are: what policy should the scout use to select nodes for his $n$ node expansions, and how should the general choose his leaf node?

For example, suppose the scout is searching the depth 5 tree whose top two levels look as pictured below after two node expansions. (Nodes A, B and C are the roots of unexpanded subtrees. The numbers beside arcs indicate expanded nodes.) Let $p = 0.85$. Suppose the scout has resources left for only *two* more node expansions. Which node should he expand first?




Or suppose the pictured tree is the situation that the (exhausted) scout reports to the general. What leaf should the general choose?

The answer to the general's dilemma is not hard. He should choose any leaf below the frontier node with least *zero value*, where the zero value of frontier node $\beta$ is

$$\text{sum of arcs on the path from the root to } \beta$$
$$+ \text{ (distance from } \beta \text{ to a leaf below it)} \times p .$$

In the tree pictured above, the general should choose any leaf below node C, whose zero-value is 4.4.

One might then wonder whether the scout should make his decisions based on the same criterion. The answer is both a reassuring "yes" and a surprising "no". Under the condition that $n$ (number of node expansions allowed) is no more than $N$ (depth of the tree), we have shown that

(1) If $p < .59$, this "greedy policy" (expand any node with least zero-value) is optimal.

(2) For any $n$, if $p$ is large enough, situations may naturally arise wherein applying the greedy policy when $n$ node expansions remain is *not* an optimal decision. The tree pictured above is one such situation. The scout should expand either node A or node B!

We have shown that the above problem can be described as a Markov decision process [3]. The proof of the first result is an induction that invokes the theory of branching processes [1] to prove a key lemma.

We have also investigated a variant of this model wherein the scout expands arcs rather than nodes, thereby learning the value of the expanded arc. We have shown how to transform this variant to a Markov decision process that is essentially the same as that for the problem described above. The analysis proceeds similarly and yields corresponding results.

We most recently studied the question

***when the greedy policy is not optimal, just how bad is it?***

The work so far suggest two conjectures for which we have proof for $p \leq .682$.

(1) The optimal decision is always more than "one away" from the greedy policy. That is, it is never optimal to expand a node whose zero-value is neither the smallest nor next-to-smallest of the frontier nodes.

(2) For fixed $p$, there is an $M$ such that if more than $M$ node expansions remain, the greedy decision is optimal. Hence, the difference between the greedy and optimal policies does not grow with the number of node expansions allowed.

The results from this investigation will form the Ph.D. thesis of David Mutchler. In 1986 he will join the Information Theory Section within the Computer Science and Systems Branch of the Naval Research Laboratory. There he will join a small team investigating questions like those listed at the beginning of this section, in the context of both AI heuristic search and classical military search.

[1] Harris, Theodore E. *The Theory of Branching Processes* (Springer-Verlag, Berlin, 1963).

[2] Karp, R.M. and Pearl, J. Searching for an optimal path in a tree with random costs. *Artificial Intelligence 21* (1983), 99-116.

[3] Ross, Sheldon M. *Introduction to Stochastic Dynamic Programming* (Academic Press, New York, 1983).

**Searching for near-optimal treatment procedures (Loveland, Lanskron).**

*Introduction.* There is an extensive literature in the binary testing problem, a problem featuring the analysis of optimal and near-optimal test procedures with respect to expected cost. The solutions are presented as decision trees. See [1] for a survey of literature on this general problem. One of the first research papers written under this grant was a paper extending the work of Garey and Graham [2] from the equal probability case to the arbitrary probability case [3]. This went a long ways towards completing our understanding of the binary splitting approximation algorithm for binary testing. While working in this area we were struck by the fact that for computer scientists, physicians and anyone else interested in repairing faults as well as finding faults this was the wrong problem. In many situations one wishes not to isolate the fault as the final solution but to treat the fault, and treatment may often occur before the fault is isolated. Indeed, the treatment can also be in part a test: "Take two aspirin and, if not better, see me in the morning". Surprisingly, no theory of tests and treatments parallel to the theory of binary testing seems to appear in the literature. We outline a model for the test-and-treatment problem and present some initial results for this model. A special case of the binary testing problem (the "complete" test case) has a simple and quickly computed solution, the Huffman coding procedure. We had hoped that there might be an interesting generalization of the Huffman procedure for the analogous case in the test-and-

treatment problem but the rich interactions between nodes, which themselves are clusters of treatments, seems to preclude a simple algorithm to solve this special case. We then discuss an approximation algorithm for the simplest case and illustrate the node interaction that makes finding simple optimal algorithms difficult. A paper presenting the full details on this material will be written in the Fall (1986).

*The model.* We first present the model for the binary testing problem because the model we consider is an extension of this binary testing model. The binary testing problem is presented by $n$ objects, $n$ *a priori* probabilities of fault, and $m$ tests with associated costs. Let $U = \{o_1, \ldots, o_n\}$ denote the $n$ objects, let $\{p_1, p_2, \ldots, p_n\}$ denote the $n$ *a priori* probabilities that report the user's estimate of the likelihood of the corresponding object being faulty, and let $\{T_1, \ldots, T_m\}$ denote the $m$ binary tests with associated costs $\{C_1, \ldots, C_m\}$.

We make various assumptions to simplify the problem analytically. We assume that there is only one faulty object, so $\sum p_i = 1$. The assumption that tests are binary means that they are reliable and unambiguous; in particular we can model a test by a subset of the universe. The test set is defined as follows: an object is placed in the test set if the test gives a positive response when that object is the faulty object. We will let $T_i$ denote the test set as well as the underlying test, because functionally they are equivalent. Previous analytical work has usually assumed that the tests all have the same cost, chosen arbitrarily as a unit cost; i.e., $C_i = 1$, all $i$.

Although this may seem draconian, much has been learned using this restriction. This knowledge serves as a message regarding the more general case with arbitrary costs. (See [2], [3]). We return to this point later.

The outcome of the problem is a decision tree that instructs one as to how to apply the tests, where the choice of test is a function of the outcome of previous tests. For any particular problem one follows a single path of the tree, branching as determined by test outcome, until the faulty object is isolated. We seek the tree of minimum expected cost, where expected cost is given by

$$EC = \sum_{i=1}^{n} Path_i \cdot p_i \tag{1}$$

with $Path_i$ defined as the sum of the costs of the tests encountered. In the case of uniform cost for tests $Path_i$ becomes the number of tests encountered.

The model for the (binary) test-and-treatment problem that we adopt here extends the binary testing model by the addition of treatments $\{T_{m+1}, \ldots, T_{m+r}\}$ with associated *a priori* probabilities $\{p_{m+1}, \ldots, p_{m+r}\}$ and associated costs $\{C_{m+1}, \ldots, C_{m+r}\}$. Our indexing convention reserves the first $m$ indices for tests and the last $r$ indices for treatments, which allow a uniform notation for both tests and treatments. Like tests, treatments are representable by subsets of $U$, but of course the meaning is quite different. If a treatment is applied then the unknown object is considered (completely) treated if it is in the treatment set, and not treated (or otherwise altered) if it is not in the treatment set. However, in each case the cost of the treatment is incurred. In the decision tree that represents a given test-and-treatment (TTr) procedure there would be only one arc below a node representing a treatment, the arc that represents the continuing path for non-treatment, i.e., when the unknown object is not in the treatment set. The procedure must treat the unknown object, so every branch of the decision tree will end in a treatment.

Our objective is still the same, to find procedures that minimizes the expected cost. Expected cost still is defined by formula (1) used for the binary testing problem, but the notion of path now changes to include treatment nodes.

Figure 1 is an example of a test-and-treatment problem presentation with two TTr procedures presented. Although we believe that Procedure 2 is optimal, the computation to establish that is sufficiently time consuming that optimality has not been proven. The decision trees have been stylized for easier reading. Although a treatment should have only one arc below it, we have added a second arc, with double lines, to record at the end of that arc the objects treated. Technically the objects treated should label the treatment made itself, the convention we follow when the treatment is at the end of a path and all objects associated with that path are treated. In general, a test or treatment labels a node, and an object with its associated *a priori* probability (alternatively, its *weight* ) labels the end of a path. The expected cost value for each procedure is also given.

A Sample Test-and-Treatment Problem

| Objects | $o_1$ | $o_2$ | $o_3$ | $o_4$ | $o_5$ |
|---|---|---|---|---|---|
| Probabilities | .3 | .3 | .2 | .1 | .1 |

Tests/Treatments

| | Name | Set | Cost |
|---|---|---|---|
| Tests | $T_1$ | {2,3} | 1 |
| | $T_2$ | {2} | 1 |
| | $T_3$ | {3,4} | 1 |
| Treatments | $T_4$ | {1,4} | 4 |
| | $T_5$ | {2,5} | 4 |
| | $T_6$ | {2,3} | 5 |
| | $T_7$ | {3} | 2 |
| | $T_8$ | {1} | 1 |

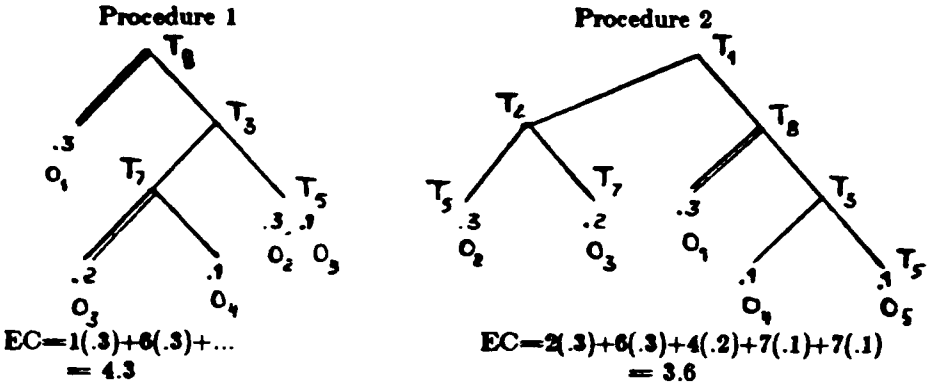



Figure 1

*Finding low-cost procedures.* Using an appropriate dynamic programming or branch-and-bound algorithm, one can determine a minimum cost TTr procedure (decision tree) even with arbitrary test costs. However, such algorithms take at least $m\,2^n$ steps to execute in the general case, where, as before, $n$ is the number of objects and $m$ is the number of tests. This exponential growth in the number of objects means that in practice only small problems can be solved exactly. Thus there has been a great interest in fast algorithms that find low-cost (but not necessarily optimal) test procedures. When algorithms are presented, the obvious questions are: (1) how fast is the algorithm? (2) how close to optimal are the resulting test procedures?

To proceed we need some definitions.

A *complete TTr problem* has a set of tests such that each subset $S$ of $U$ is both a test set and a treatment set. (For tests it actually suffices that either $S$ or $U-S$ be a test, by symmetry for tests.) The *complete testing problem* is a restriction of the complete TTr problem to test sets.

A complete TTr problem (or testing problem) is an important subcase because it is assured that whenever a test (treatment) is desired, it exists and may be used. The importance of this subcase is documented by some major properties for the binary testing procedure, which we now state. See [2] and [3] for details and further properties.

The following hold for the binary testing problem with unit test costs.

I. There is a $O(n \log n)$ algorithm to find the optimal test procedure for the (unit cost) complete testing

problem.

II. The incomplete testing problem is *NP*-hard. (That is, all evidence is that such problems cannot all be solved in $0(n^k)$ steps for any given integer $k$.)

III. For the incomplete testing problem the most natural fast approximation algorithm has been evaluated regarding its speed (easily seen as $0(mn)$) and its expected cost approximation to optimal. See [2] and [3].

With this background in mind, we undertook the study of the test-and-treatment problem, which as mentioned earlier, seems to be the more correct problem statement for most real-life situations. Our original goals were:

A. To find a fast optimal algorithm for the complete TTr problem (for a restricted cost case);

B. For the incomplete TTr problem to find a good approximation algorithm.

After considerable study on the former question we have reason to doubt the interest, perhaps even the feasibility, of our first goal. Besides giving some quite restricted results we will demonstrate why seeking an optimal solution may not be worthwhile except in the restricted case we mention. (We have to date done limited work on the second goal; that work is beyond the scope of this summary.)

By an *equiprobable TTr problem* we mean any *TTr* problem where all *a priori* probabilities have equal value, i.e., $p_i = 1/n$, where $n$ is the number of objects in $U$.

We now consider briefly a dynamic programming solution to the general TTr problem. For any subset $S$ of $U$, we define $EC(S)$ by

$$EC(S) = \min\{\min_{1 \le i \le m}(C_i \cdot |S| + EC(S \cap T_i) + EC(S - T_i)), \quad (2)$$

$$\min_{m < i \le m+r}(C_i \cdot |S| + EC(S - T_i))\},$$

where $|S|$ denotes the sum of the weights of the objects in set $S$, $C_i$ denotes the cost of test or treatment $i$, $EC|\phi| = 0$, and any term reducing to $EC(S)$ itself on the right is undefined. In general the cost of computing $EC(U)$, the desired answer, is exponential in $n$ because there are $2^n$ subsets that need consideration. Our colleague Robert Wagner observed that if the partial expected cost $EC(S)$ only depends on the cardinality of $S$ then this minimization can be solved in $0(n^2)$ steps. Basically, this is because one needs to know only $EC(\#S)$ rather than $EC(S)$ for all smaller subsets $S$. ($\#S$ denotes the cardinality of $S$). This special case can be realized for the equiprobable complete TTr problem with all test costs the same and treatment costs proportional to the weight of the treatment set, for example. (See [4] for more details. A method of parallel computation of the general dynamic programming formulation (2) for the TTr problem is presented in [5].)

Before proceeding to discuss an approximation algorithm for a special case we should note that the most obvious special case is not interesting. The complete TTr problem that has all treatments as well as all tests with unit cost is clearly easy to solve: simply invoke the universal treatment (that treatment with treatment set U) so that every object is treated by that one treatment. That gives an expected cost of 1. Clearly, any other procedure is more costly. For uniform treatment costs other than unit cost (the cost of each test) the answer remains the same because it costs as much to treat a subset of $U$ as to treat all of $U$.

Some experimental work has shown us that we often do quite well in an arbitrary TTr problem (including the incomplete TTr problem case) if we choose the treatment with the lowest cost/power ratio and invoke that treatment, and recurse on that strategyy. By power we simply mean the weight of the treatment set. This simple rule fails if a number of treatments have nearly the same cost/power ratio. The example in Figure 1 has all the treatments in the problem except the last with a cost/power ratio of

10; and this makes the outcome more difficult to ascertain. Treatment $T_8$ has cost/power ratio less than 4, so by our guideline should be favored over the other treatments. Both the procedure illustrated have $T_8$ near the top of the decision tree, and the reader may wish to verify that placing other treatments before $T_8$ yields worse procedures. The example procedure does illustrate that one may want to use tests before invoking the most effective treatment for best expected cost.

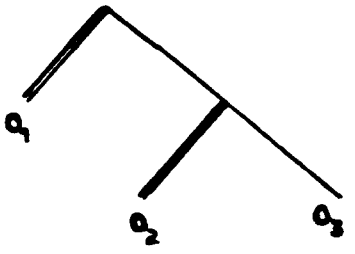
Because the single most important determiner of value for treatments seems to be the cost/power ratio, our special case investigations have focused first on the subcase where all treatments have the same cost/power ratio. We hereafter denote that ratio by $k$. Test costs will be fixed at unit cost. Again, we demand that all tests and treatments be present.

We have determined certain properties of optimal procedures for this special case.

*Lemma 1.* No non-singleton treatment appears in an optimal procedure for this subcase.

*Lemma 2.* All tests occur prior to treatments.

We omit all proofs although the proof of Lemma 1 follows very quickly from the nature of expected cost computations in this special subcase. Any multiobject treatment can be replaced by a sequence of singleton treatments for the same objects with lower expected cost, which we call a *cascade*. For example, the single treatment given by treatment set $\{o_1,o_2,o_3\}$ can be replaced by the cascade of Figure 2 with the expected cost then reduced.




A cascade
Figure 2

We now give an approximation algorithm for a yet more restricted subcase, namely, the subcase under consideration (complete TTr problem, unit cost tests, treatments of cost $kw$ where $w$ is the weight of the treatment set) plus the equiprobable requirement. We noted that for the special case the dynamic programming method allowed computation of the optimal decision tree in $O(n^2)$ steps. The approximation algorithm gives a decision tree in time essentially independent of $n$ and $k$ (constant time). We present a bound on the relative error, a result that is non-trivial, requiring considerable understanding of the nature of low-cost decision trees for this problem. We will close the paper with an attempt to illustrate why a fast optimal algorithm (beyond the special case dynamic programming algorithm) is likely to be complex and not likely to be found soon, if it exists. Also the problem of finding approximation algorithms for less restricted classes must take these concerns into consideration.

The decision tree class to be used for this special case is a simple class. The trees have the following properties:

a) the superleaves of the tree are all at the same level, where each superleaf is a cascade (so the position of the superleaf locates the root of the cascade);

b) the cascades differ by at most one in the number of objects treated;

c) the level of occurrence of the superleaves is level $l$ (the rest is level 0) where $l$ is the least integer $z$ such that

$$k \leq 4 \cdot 2^z .$$

Thus all tests, and only tests, occur above level $l$ with nearly identically formed cascades beginning at level $l$. The cost/power ratio $k$ determines the transition level.

We shall call the class just defined the class of *level l* procedures. See Figure 3 for examples of level $l$ procedures.

An upper bound that holds for all $k$ is given by

$$EC_l - \frac{opt}{EC_l} < 1/l \tag{3}$$

where $EC_l$ is the expected cost of the above mentioned decision tree and opt is the minimal expected cost possible. Experimentation shows that for small $n$ the approximation is actually much better than the upper bound suggests. The actual relative error does not seem to improve with $n$ for a fixed $k$, and also varies considerably with $k$ even for small values of $k$. The upper bound is as weak as it is partly because it represents all values of $k$. The theorem statement below helps explain the varying relative error for small $k$.

The relative error result follows from a key theorem that holds for this special case.

*Theorem.* For every $n > 0$ and every $l > 0$, there is a cost/power ratio value $k$ such that the $n$-object level $l$ procedure is an optimal procedure for that value of $k$.

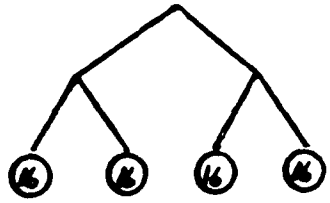
One should note that for each $n$ and $l$ there is only one $n$-object level $l$ decision tree modulo the left-right orientation of branches of binary trees. The theorem states that this tree is optimal for some $k$. We can determine some of these $k$ values but the expression is messy. At intermediate $k$ values the approximation seems quite good but is hard to characterize analytically. Thus our relatively modest upper bound.

What is at least as interesting as this upper bound is a characteristic of optimal trees that we can hint at by example. Although there is symmetry to the problem presentation (equiprobable weights, costs uniform for tests and dependence only on the number of objects in treatments) the optimal tree is not necessarily fully symmetric, due to what we call "migration of objects" from cascade to cascade as $k$ changes. Without going into the specific analysis, we demonstrate the effect in Figure 3 where we present three decision trees for a specific TTr problem.

For Figure 3 we choose $n = 64$ and $k = 16$, which by our formula for determining the level $l$, $l = \min z\,(k \leq 4 \cdot 2^z)$, puts $l = 2$ by virtue of the equal sign; had $k = 16.001$ then $l = 3$ would be needed. That is reflected by the same expected cost value for the level 2 tree and the level 3 tree. (The circle with enclosed number represents the number of elements in a cascade; we chose $n$ so that all cascades are equally populated to remove the "excess objects" effect. The branching above the superleaves represents tests that split the relevant sets of objects exactly in half.)

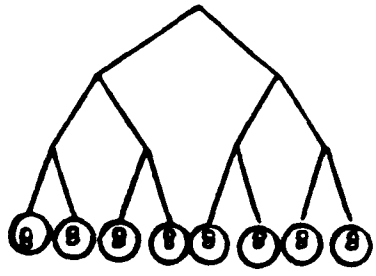
Level $l$ procedures are not always optimal

Example: $n = 64$ $k = 16$




level 2 procedure: 16 obj/cascade

$$EC_2 = \frac{1}{64}[l \cdot 64 + \frac{k}{64}(4 \cdot \frac{s(s+1)}{2})]$$
$$= \frac{1}{64}[2 \cdot 64 + \frac{1}{4}(4 \cdot 136)] = \frac{1}{64}(264)$$




level 3 procedure: 8 obj/cascade

$$EC_3 = \frac{1}{64}[3 \cdot 64 + \frac{1}{4}(8 \cdot 36)] = \frac{1}{64}(264)$$

level 2,3 tree:

$$EC_{2,3} = \frac{1}{64}[2 \cdot 28 + 3 \cdot 36$$
$$+ \frac{16}{64}(4 \cdot 45 + 2 \cdot 105)]$$
$$= \frac{1}{64}[56 + 108 + 45 + 52.5]$$
$$= \frac{1}{64}[261.5]$$

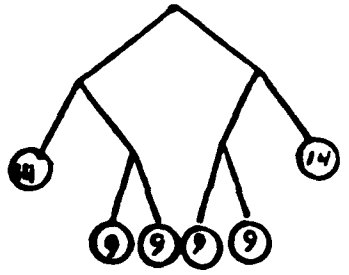



Figure 3

The third tree of Figure 3 has cascades with roots at levels 2 and 3 so is not a level $l$ procedure. Yet its expected cost is lower than the two level $l$ trees. One might have expected, even hoped, at the transition value of $k$ (viewing $k$ as increasing) where a level 2 tree passes to a level 3 tree, that an even splitting of each cascade from 16 members to two cascades of 8 members each would yield optimal trees. We see by example that instead the 8-member cascades absorb an extra member and allow a 14-member cascade which is not "big enough" to split at that $k$ value. This is what we term "migration". This migration can be characterised but the resulting algebra makes computation very messy and the determination of an algorithm for finding optimal trees very unpleasant, if doable, even for this simple case. It is better to use the dynamic programming formulation in this special case if optimal trees are needed, and in general we surely will settle for approximate solutions, even in the complete TTr problem case. (Recall that the incomplete TTr problem is NP-hard anyway, since the simpler incomplete testing problem is NP-hard.)

We now state briefly how the terms in the expected cost are computed in Figure 3. The first line in the computation of $EC_1$, the expected cost for the level 2 procedure outlines the computation symbolically. We do not sum the terms $Path_i \cdot p_i$ directly but aggregate components. First we factor out the common weight $p_i$ (i.e. 1/64) so we need only detemine $Path_i$. The tests, of unit cost, cost $l$ units for each of the 64 objects. The cascade is composed of unit treatment costs of k/64, and there is one less object subject to each sequential treatment so for $s$ objects in a cascade there are $\sum_{i=1}^{s} i$ object-treatments. (Compare with $(ks/64) \cdot s$ for a single treatment for all $s$ objects.)

We presently have a model for reasonable approximate procedures in the arbitrary weight case but no upper bound or relative error yet. Such models may serve as well for the incomplete case. Finding

bounds on their relative error is another matter however.

The integrated theory of test-and-treatment procedure design is clearly of interest and it is our hope eventually to better understand how to find, with reasonable effort, good low-cost procedures for accomplishing this task. We are hopeful that at least for the complete TTr we can do well for the cost structure outlined here.

## References

[1] Payne, R.W. and D.A. Pierce. Identification keys and diagnostic tables: a review. *J. Royal Stat. Soc. (Series A) 143*, 253-292, 1980.

[2] Garey, M.R., Graham, R.L. Performance bounds on the splitting algorithm for binary testing. *Acta Infor 3*, 347-355, 1976.

[3] Loveland, D.W. Performance bounds for binary testing with arbitrary weights. *Acta Infor 22*, 101-114, 1985.

[4] Wagner, R.A. A polynomial time algorithm for the complete test-and-treatment decision tree problem. C.S. Report, Comp. Sci. Dept., Duke University, 1986.

[5] Duval, L.D., R.A. Wagner, Y. Han and D.W. Loveland. Finding test-and-treatment procedures using parallel computation. *Proc. of the 1986 Int'l Conf. on Parallel Processing*, St. Charles IL, August, 1986.

**Boolean Vector Machine (Wagner, Han).**

This report covers the partial support received from the Air Force during the past three years.

The principal accomplishments during this period are represented by two papers. The first, jointly written by the two people above, and D.W. Loveland, is to be presented in August, 1986, at the International Conference on Parallel Processing [1]. That paper reports on a new parallel algorithm for computing optimal test and treatment decision trees. On a sufficiently large Boolean Vector Machine, the algorithm promises substantial speedup over any existing sequential algorithm, for the general problem. By the general problem, we mean the test and treatment problem given an arbitrary effective set for each test and treatment, for which that test or treatment succeeds. In perspective, the algorithm presented there is adequate when relatively few distinct tests and treatments are given. For a number of tests which is exponential in the size of the object-space, the algorithm demands too many PE's to be useful (unless we find funds for a one billion PE machine). However, the algorithm was developed in sufficient detail to ensure that the small per-PE memory of the BVM imposes no limitation on the algorithm, and that the algorithm's logical correctness, and timing analyses are both correct. The version of this paper presented at the Parallel Processing Conference is necessarily very short. The full paper [2] has been submitted for publication to IEEE Trans. on Computers.

A second result, obtained very recently, also applies to the test and treatment problem, but this time to an important special case only. An algorithm was obtained for the case in which all possible test and treatment sets are available, and in which the a priori probabilities that each given object be the distinguished object are all equal. Variants of the problem solved by this method include differences in the cost function associated with tests $O(n^2)$, where there are $n$ objects in the object-space. Previously, the best algorithm known for this case required time $O(4^n)$. A paper [3] describing this result has been written, and will be submitted to a journal, probably J. ACM, after one additional revision pass.

As reported last year, some work was done by Wagner on the Boolean Vector Machine hardware design. This work was needed to correct a problem in the implementation of the on-chip memory of that design, and has proven successful. As of March 10, 1986, an 8-chip BVM system has been running at UNC. The system contains 512 Processing Elements, and runs at 2.5 MHz instruction rate. A production run of the chips it is made of has been received and tested; those chips test individually at over 4 MHz instruction rate, and some 40% of the chips proved to be correct at that speed. The system itself would run at 4 MHz, except for lack of funds to improve the system clock timing slightly – the clock generation circuit assumes that the 4-phase clock for the system uses equal-duration clock phases, while the ideal clock durations accepted by the chips varies somewhat from equal-duration. Thus, I conclude that a system running at 4 MHz is actually constructable from the parts we have built, and that therefore the timing claims we have made are correct. If more funds are obtained for the purpose, we can rather easily build a 32-chip system, running at or very close to the design-target 4 MHz instruction rate.

References


1. Duval, D., Wagner, R.A., Han. Y., and Loveland, D.W. Finding Test-and-Treatment Procedures Using Parallel Computation. To appear in *Proc. 1986 Int'l. Conf. on Parallel Processing.*
2. Duval, D., Wagner, R.A., Han, Y., and Loveland, D.W. Finding Test-and-Treatment Procedures Using Parallel Computation. Dept. of Computer Science, Tech Report. CS-1986-5, Oct. 1985. Submitted to *IEEE Trans. on Computers.*
3. Wagner, R.A. A Polynomial Time Algorithm for the Complete Test-and-Treatment Decision Tree Problem. Dept. of Computer Science, Duke University. CS-1986-24. July 1986.


**Supported Personnel**

| | |
|---|---|
| A. Biermann | (Co-Principal Investigator) |
| A. Fahmy | |
| P. Fink | (Ph.D. dissertation, 1983) |
| K. Gilbert | |
| Y. Han | |
| P. Lanzkron | |
| D. Loveland | (Principal Investigator) |
| D. Mutchler | (Ph.D. dissertation, 1986) |
| A. Reibman | |
| M. Valtorta | (Ph.D. dissertation, due 1987) |
| R. Wagner | (Co-P.I.) |

Students without dissertation credit either worked as research assistants and then chose other work for thesis work or are still not close enough to concluding a dissertation to warrant an explicit marking.

## Publications and Theses

July, 1981 - June, 1986

Chronologically ordered


1. Biermann, A., J. Fairfield and T. Beres. Signature tables and learning. *IEEE Trans. on Systems, Man and Cybernetics*. Oct. 1982, pp. 635-648..

2. Biermann, A. Dealing with search. *Automatic Program Construction Techniques* (Eds. Biermann, Guiho, Kodratoff). MacMillan Publ. Co., 1984.

3. Ballard, B. The *-minimax search procedure for trees containing chance nodes. *Artif. Intelligence*, Oct. 1983, pp. 327-350.

4. Loveland, D. Performance bounds for binary testing with arbitrary weights. *Acta Informatica* 22(1985), pp.101-114.

5. Ballard, B. Non-minimax search strategies for minimax trees: theoretical foundations and empirical studies. Duke C.S. report CS-1983-13, July, 1983. (submitted for publication)

6. Reibman, A., B. Ballard. Non-minimax search strategies for use against fallible opponents. *Third Nat'l. Conf. on Artif. Intell.*, Washington, D.C., August, 1983. Conditionally accepted in *Artif. Intelligence*.

7. Fink, P. The acquisition and use of dialogue expectation in speech recognition. Ph.D. Thesis, Computer Science Department, Duke University, 1983. Also C.S. Report CS-1983-101.

8. Jackoway, G. Associative networks on a massively parallel computer. A.M. Thesis, Computer Science Department, Duke University, 1984.

9. Ballard, B., N. Tinkham. A phrase-structured grammatical framework for transportable natural language processors. *Amer. J. Comput. Linguistics*, 1985, pp. 81-96.

10. Valtorta, M. A result on the computational complexity of heuristic estimates for the A* algorithm. *Infor. Sciences*, 34, 1984, pp. 47-59.

11. Biermann, A.W. Automatic programming: a tutorial on formal methodologies. *Jour. of Symbolic Comp. 1*, 1985, pp. 119-142.

12. Fink, P.K., A.W. Biermann. The correction of ill-formed input using history based expectation with applications. *Computational Linguistics, 12*, Jan.-Mar. 1986, pp. 13-36.

13 Loveland, D.W. Finding critical sets. Accepted in *Journal of Algorithms*.

14. Mutchler, D.C. Search with very limited resources. Ph.D. thesis, Computer Science Dept., Duke University, 1986.

15. Duval, L.D., R.A. Wagner, Y. Han, and D.W. Loveland. Finding Test-and-Treatment procedures using parallel computations. Submitted to *IEEE Trans. on Computing*.

Note: Bruce Ballard, while not funded by this grant this year, was funded last year, and in previous years on an earlier AFOSR grant.

## Papers in Preparation

Biermann, A., K. Gilbert, A. Fahmy, B. Koster, "On the Errors that Learning Machines Will Make," in preparation.

Loveland, D.W. and P. Lanzkron.

Wagner, R.A. A Polynomial Time Algorithm for the Complete Test-and-Treatment Decision Tree Problem. Dept. of Computer Science, Duke University. CS-1986-24. July 1986.

## Conference Presentations

July, 1981 - July, 1986

1. Ballard, B. A search procedure for perfect information games of chance. *Second National Conf. on Artif. Intell. -82*, Pittsburgh, Aug. 1982.

2. Loveland, D. Knowledge acquisition and evaluation. Army Conf. on AI Application to Battlefield Info. Systems, Silver Springs, Md., April 1983.

3. Ballard, B.W. and A.L. Reibman. What's wrong with minimax? *1983 Conf. on Artif. Intell.* Rochester, Mich., April 1983.

4. Reibman, A.L. and B.W. Ballard. Non-minimax search strategies for use against fallible opponents. *21st ACM Southeast Region Conf.*, Durham, NC, April 1983.

5. Reibman, A.L. and B.W. Ballard. Non-minimax search strategies for use against fallible opponents. *Third Natl. Conf. on Artif. Intell.*, Washington, D.C., August, 1983.

6. Loveland, D.W. and M. Valtorta. Detecting ambiguity: an example of knowledge evaluation. *Eighth Intern. Joint Conf. on Artif. Intell.*, Karlsruhe, W. Germany, August, 1983.

7. Valtorta, M. A result on the computational complexity of heuristics for the A* algorithm. *Eighth Intern. Joint Conf. on Artif. Intell.*, Karlsruhe, W. Germany, August, 1983.

8. Ballard, B. The syntax and semantics of user-defined modifiers in a transportable natural language processor. *Proc. of the 22nd Annual Meeting of the Assoc. Comput. Linguistics*, Stanfod University, July, 1984.

9. Valtorta, M. Knowledge refinement in rule bases for expert systems: an application-driven approach. *First Intern. Workshop on Expert Database Systems*, Kiawah Island, S.C., October, 1984.

10. Valtorta, M., B. Smith and D. W. Loveland. The Graduate Course Adviser: a multi-phase rule-based expert system. *IEEE Workshop on Principles of Knowledge-Based Systems*. Denver, December, 1984.

11. Biermann, A., K.C. Gilbert, A. Fahmy and B. Koster. On the errors that learning machines will make. *Fourth Army Conf. on Applied Math. and Computing*. Ithaca, May 1986.

12. Loveland, D.W. Introducing treatments into test procedures. *Fourth Army Conf. on Applied Math. and Computing*, Ithaca, May 1986.

13. Duval, L.D., R.A. Wagner, Y. Han, D.W. Loveland. Finding Test-and-treatment procedures using parallel computation. *1986 Int'l. Conf. on Parallel Proc.*, St. Charles, IL. August 1986.

Note: Bruce Ballard, while not funded by this grant this year, was funded last year, and in previous years on an earlier AFOSR grant.

END

4-88

DTIC